OAEP is Secure Under Key-dependent Messages

Michael Backes[1,2], Markus Dürmuth[1] and Dominique Unruh[1]

[1] Saarland University, Saarbrücken, Germany, {backes,duermuth,unruh}@cs.uni-sb.de
[2] Max-Planck-Institute for Software Systems, Saarbrücken, Germany, backes@mpi-sws.mpg.de

February 19, 2008

Abstract. Key-dependent message security, short KDM security, was introduced by Black, Rogaway and Shrimpton to address the case where key cycles occur among encryptions, e.g., a key is encrypted with itself. We extend this definition to include the cases of adaptive corruptions and arbitrary active attacks, called adKDM security incorporating several novel design choices and substantially differing from prior definitions for public-key security. We also show that the OAEP encryption scheme (using a partial-domain one-way function) satisfies the strong notion of adKDM security. The OAEP construction thus constitutes a suitable candidate for implementating symbolic abstractions of encryption schemes in a computationally sound manner under active adversaries.

Keywords: Key-dependent message security, chosen ciphertext attacks, RSA-OAEP.

1 Introduction

Encryption schemes constitute the oldest and arguably the most important cryptographic primitive. Their security was rigorously studied very early, starting with Shannon's work for the information-theoretic case [29]. Computational definitions for public-key encryption were developed over time, in particular in [21,30,28,17]. For symmetric encryption, the first real definitions were, to the best of our knowledge, given in [17,26,7], using the same basic ideas as in public-key encryption. While these definitions seemed to take care of standard usage of encryption schemes, it was soon recognized that larger protocols might pose additional requirements on the encryption schemes, e.g., in multi-party computations with dynamic corruptions as in [6]. It was also recognized that in some cases, symmetric encryption initially seemed to be the appropriate method to use, but upon study other primitives such pseudorandom permutations [9,7] or authenticated encryption [11,8] proved to be better.

A specific additional requirement some larger protocols pose on encryption schemes is the ability to securely encrypt key-dependent messages. One speaks of key-dependent messages if a key K is used to encrypt a message m where m contains or depends on the key K (or the corresponding secret key in the case of public-key encryption). The first concrete use of this case seems to have been in [13], where multiple private keys were used to encrypt one another in order to implement an all-or-nothing property in a credential system to discourage people from transferring individual credentials. Such key cycles also occur in implementations of disk encryption in, e.g., Windows Vista, that can store an encryption of its own secret keys to the disk in some situations. Key cycles also occur in some naively designed key exchange protocols of session keys given master keys shared among the two parties or with a key distribution center, where at the end of the protocol the newly exchanged key is “confirmed” by using it to encrypt or authenticate something that might include the master keys.

Another area that has brought additional requirements on cryptographic primitives, and in particular that of encryption with key cycles, is the use of formal methods or “symbolic cryptography”. Here the question is whether simple abstractions of cryptographic primitives exist that can be used by automated proof tools (model checkers or theorem provers) to prove or disprove a wide range of security protocols that use cryptography in a blackbox manner. The original abstractions used by this automation community are term algebras constructed from certain base types and cryptographic operators such as E and D for encryption and decryption. They are often called Dolev-Yao models after the first such abstraction [18]. As soon as one has a multi-user variant of such a model, the keys are terms, and from the term algebra side it is natural that keys can also be encrypted, i.e., most models simply assume that key cycles are allowed. Once cryptographic justification of such models was started in [2], it was recognized that key cycles had to be excluded from the original models to get cryptographic results. The same holds for later results [1,24,5,25,27,3,16,15].

Motivated primarily by symbolic cryptography, a definition of key-dependent message security (*KDM security*) was introduced in [12]. It generalizes the definition from [13] by allowing arbitrary functions of the keys (and not just individual keys) as plaintexts, and by considering symmetric encryption schemes. In [4] it was shown that, in the case of symmetric encryption, an extension of the KDM definition that additionally allows for a limited revelation of secret keys of honest users, called DKDM security, is suitable for extending results about the justification of Dolev-Yao models to include protocols with key cycles. Security in the presence of key-dependent messages has so far only been achieved in the random oracle model.[1] Extensions of KDM security for public-key encryption to active adversaries have not been proposed yet, and the establishment of meaningful definitions for this case indeed raises non-trivial problems.

Our Contributions. We first propose a new definition of security under key-dependent messages, called *adKDM security*, that captures security against active attackers and adaptive corruptions in the case of public-key encryption. This definition incorporates several novel design choices and substantially differs from prior definitions for public-key security; in particular, it allows the adversary to iteratively construct nested encryptions without necessarily revealing inner encryptions, and it is required to keep track of the knowledge that the adversary maintains in an ideal setting.

We then investigate the OAEP encryption scheme and prove that it satisfies adKDM security in the random oracle model, assuming the partial-domain one-wayness of the underlying trapdoor-permutation. This in particular shows the OAEP construction to constitute a suitable candidate for soundly implementing symbolic abstractions of cryptography.

The need to incorporate key dependencies and the adaptive nature of adKDM security require substantial changes to the CCA2-security proof of OAEP. In particular, adKDM

[1] In [22] and [23], the problem of implementing KDM secure symmetric encryption schemes without random oracles is investigated. There, solutions are given for relaxed variants of KDM security, e.g., security against a bounded number of queries or security with respect to a *single* key dependency function. No scheme is known, however, that fulfills any form of full-fledged KDM security (passive or active) without the use of random oracles.

security does not allow for determining in advance which encryptions will be used as challenge encryptions. At the point of construction of these bitstrings, the adversary might not even know the challenge encryptions. Consequently, performing the reduction to the underlying assumption requires us to lazily construct them in order to decide as late as possible which encryption constitutes a challenge encryption.

2 Preliminaries

In this section, we present some definitions and conventions that will be used later on in the paper.

Notation. Let $\oplus$ denote the XOR operation, and let $\|$ denote concatenation. For a probabilistic algorithm B, let $y \leftarrow B(x)$ denote assigning the output of $B(x)$ to y. Let $\Pr[\pi : X]$ denote the probability that π holds after executing the instructions in X (which are of the form $y \leftarrow B(x)$). A function in n is negligible if it is in $n^{-O(1)}$. A function is non-negligible if it is not negligible. We formulate all our results for uniform adversaries, but they hold for nonuniform adversaries as well.

Definition 1 (Circuit). *A* circuit *is a Boolean circuit with $n_1+\dots+n_t$ input bits ($t \geq 0$) and m output bits. The circuit may have arbitrary fan-in and fan-out,* AND-*,* OR- *and* NOT-*gates, and—in the case of an encryption scheme in the random-oracle model—gates for querying the random oracle(s). We assume that a circuit is always encoded by explicitly specifying all its gates and the numbers $n_1, \dots, n_t, m$. The evaluation $f(x_1, \dots, x_t)$ of a circuit f on bitstrings $x_1, \dots, x_t$ is defined as follows: Let x'_i be the result of truncating or padding x_i with 0^* to the length n_i. Then $f(x_1, \dots, x_t)$ is the result of evaluating f with input $x'_1\| \dots \|x'_t$.*[2]

Convention: Encryption is length-regular. For any encryption scheme, we impose the following assumption on the output of the encryption function Enc and the decryption function Dec: The length of the output of Enc depends only on the public key and the length of the message. The length of the output of Dec depends only on the *public* key and the length of the ciphertext. This can easily be achieved by suitable padding and encoding.

The OAEP scheme. The optimal asymmetric encryption padding (OAEP) scheme [10] constitutes a widely employed encryption scheme in the random oracle model based on a trapdoor 1-1 function.

Definition 2 (OAEP). *Let k denote the security parameter and let k_0 and k_1 be functions such that $k_0, k_1, k-k_0-k_1$ are superlogarithmic. Assume a 1-1 trapdoor function f with domain $\{0,1\}^k = \{0,1\}^{k-k_0} \times \{0,1\}^{k_0}$. Let $G : \{0,1\}^{k_0} \to \{0,1\}^{k-k_0}$ and $H : \{0,1\}^{k-k_0} \to \{0,1\}^{k_0}$ denote random oracles. The public and secret key for the OAEP encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ consists of a public key and a trapdoor for f. An encryption $c = \mathrm{Enc}(pk, m)$ with $|m| = k - k_0 - k_1$ is computed as $r \leftarrow \{0,1\}^{k_0}$, $s := (m\|0^{k_1}) \oplus G(r)$, $t := r \oplus H(s)$, $c := f_{pk}(s\|t)$.*

[2] Not granting a circuit access to the length of its arguments is not a restriction in our case, since this length will always be known in advance.

A decryption $\mathrm{Dec}(sk, c)$ *is computed as* $s \| t := f_{sk}^{-1}(c)$, $r := t \oplus H(s)$, $m \| z := s \oplus G(r)$ *with* $|s| = k - k_0$, $|t| = k_0$, $|m| = k - k_0 - k_1$ *and* $|z| = k_1$. *If* $z = 0^{k_1}$, *the plaintext* m *is returned, otherwise the decryption fails with output* $\perp$.

It has been shown in [19] that the OAEP scheme is IND-CCA2 secure in the random oracle model under the assumption that f fulfills the following Definition 3 of partial-domain one-wayness. They further showed that the RSA-trapdoor permutation, which is most commonly used for the OAEP scheme, is partial-domain one-way.

Definition 3 (Partial-Domain One-Wayness). *A 1-1 function* $f\colon S \times T \to \mathrm{range}\, f$ *with key generation* KeyGen_f *is* partial-domain one-way *if for any polynomial-time adversary* A *we have that*

$$\Pr\bigl[s = s' : pk \leftarrow \mathrm{KeyGen}_f, (s,t) \overset{\$}{\leftarrow} S \times T, s' \leftarrow A(pk, f_{pk}(s \parallel t))\bigr]$$

is negligible in k, *where* $A, \mathrm{KeyGen}_f, f, S, T$ *depend on the the security parameter* k. *We sometimes call this probability the* advantage *of* A.

3 The Definition of adKDM

We now present our definition of adKDM security. Since this definition incorporates several novel design choices and substantially differs from prior security definitions for public key security, we do not immediately present the definition. Instead, we start with a direct adaption of an existing definition and show using an example why this adaption is not sufficient. We proceed with several plausible approaches for extending this adaption and explain why they fail. We finally present our definition of adKDM security and explain why it solves the problems observed with the tentative definitions discussed before.

Extending DKDM security. In [4] the security notion DKDM was proposed for the case of symmetric key-dependent encryptions. It is the strongest notion of KDM security considered so far; restating it one-to-one in the public-key setting would yield the following definition:[3]

Definition 4 (DKDM, public key setting – sketch). *The DKDM oracle maintains a sequence of key pairs* pk_i, sk_i *and a random challenge bit* b. *It answers to the following queries:*

- *pk*(j)*: Return* pk_j.
- *reveal*(j) *where* j *has not been used in an enc*$(j,\cdot)$ *query: Return* sk_j.
- *enc*(j, f) *where* f *is a circuit and* j *has not been used in a reveal*(j) *query: Compute* $m_0 := f(sk_1, sk_2, \dots)$, $m_1 := 0^{|m_0|}$ *and encrypt* $c := \mathrm{Enc}(pk_j, m_b)$. *Return* c.
- *dec*(j, c) *where* c *has not been returned by an enc*$(j,\cdot)$ *query with the same key index* j*: Return* $\mathrm{Dec}(sk_j, c)$.

A public key encryption scheme $(\mathrm{Enc}, \mathrm{Dec})$ *is DKDM secure if no polynomial time adversary interacting with the DKDM-oracle guesses* b *with probability non-negligibly greater than* $\frac{1}{2}$.

[3] We have omitted one condition of their definition, namely that it should not be possible to generate a valid ciphertext without the knowledge of the secret key. This condition is not applicable to the public-key setting.

This definition is an almost immediate generalization of the IND-CCA definition to the multi-session setting (i.e., with several key pairs instead of only one). DKDM extends IND-CCA in two ways: First, the messages that are contained an $enc(\cdot,\cdot)$ encryption query may depend on all secret keys in the system. Second, one can reveal secret keys as long as the corresponding public keys have not been used for encrypting (otherwise one could decrypt a challenge ciphertext so that the definition cannot be met).

Although the notion of DKDM has been shown to be useful for soundness results for a specific class of protocols, it has obvious restrictions on the class of protocols considered. In particular, it is not allowed to reveal a key that has been used for encryption. The following simple protocol illustrates that this indeed constitutes a restriction: Alice holds two secret keys sk_1, sk_2 and a secret message m and sends the following messages to Bob:

$$c_1 := \mathrm{Enc}(pk_1, \mathrm{Enc}(pk_2, m\|sk_1\|sk_2)), \quad c_2 := \mathrm{Enc}(pk_2, \mathrm{Enc}(pk_1, m\|sk_1\|sk_2))$$

Then Bob chooses a value $i = 1, 2$ and Alice sends sk_i to Bob. We would intuitively expect the message m to stay secret since Bob learns at most one of the keys sk_1, sk_2. However, a direct reduction against DKDM security fails. Namely, we have basically three possibilities to construct the messages c_1, c_2 by querying the DKDM oracle (note that *enc* denotes the query to the adKDM oracle while Enc is the encryption algorithm):

(i) $c_1 := enc(1, g_1)$, $c_2 := enc(2, g_2)$ where g_1 and g_2 are circuits computing $\mathrm{Enc}(pk_2, m\|sk_1\|sk_2)$ and $\mathrm{Enc}(pk_1, m\|sk_1\|sk_2)$, respectively (given input (sk_1, sk_2)).

(ii) $c_1 := \mathrm{Enc}(pk_1, enc(2, g))$, $c_2 := \mathrm{Enc}(pk_2, enc(1, g))$ where g computes $m\|sk_1\|sk_2$.

(iii) $c_1 := \mathrm{Enc}(pk_1, enc(2, g))$, $c_2 := enc(2, g_2)$ where g and g_2 are as before.

(iv) $c_1 := enc(1, g_1)$, $c_2 := \mathrm{Enc}(pk_2, enc(1, g))$, where g and g_1 are as before.

Then, depending on the value of i chosen by Bob, we have to issue $reveal(i)$. In cases (i) and (ii), no reveal query is allowed since queries of the forms $enc(1,\cdot)$ and $enc(2,\cdot)$ have been performed which excludes reveal queries $reveal(1)$ and $reveal(2)$ by Definition 4. Similarly, in case (iii) we are not allowed to query $reveal(2)$, and in case (iv) we are not allowed to query $reveal(1)$. Thus in order to perform the first step, we have to know in advance what the value of i will be and to construct c_1, c_2 as in case (iii) or (iv), respectively. Of course, in the present example it is possible to save the reduction proof by guessing i; however, it is easy to thwart this possibility by performing many such games in parallel.[4] A natural approach to extend the definition of DKDM to this case would be to allow to even reveal keys sk_j that are used in encryption queries $enc(j,\cdot)$. However, a query $enc(j,\cdot)$ returns an encryption c of the message m_b. So given the secret key sk_j, we could easily determine m_b from c and therefore the challenge bit b. Therefore, we will have to distinguish between two types of encryption queries: A normal encryption query $enc(j, f)$ will return the encryption of $m_0 := f(sk_1, \dots)$ irrespective of the value of b. A challenge encryption query $challenge(j, f)$ returns m_b where m_0 is as for $enc(j, f)$ and $m_1 := 0^{|m_0|}$. This leads to the following tentative definition:

[4] E.g., Alice sends $m_1^{(1)}, m_2^{(1)}, \dots, m_1^{(n)}, m_2^{(n)}$ with $m_1^{(\mu)} := \mathrm{Enc}(pk_1^{(\mu)}, \mathrm{Enc}(pk_2^{(\mu)}, m\|keys))$, $m_2^{(\mu)} := \mathrm{Enc}(pk_2^{(\mu)}, \mathrm{Enc}(pk_1^{(\mu)}, m\|keys))$ and $keys := sk_1^{(1)}\|sk_2^{(1)}\|\dots\|sk_1^{(n)}\|sk_2^{(n)}$. Then Bob chooses $i_1, \dots, i_n \in \{1, 2\}$ and Alice sends $sk_{i_1}^{(1)}, \dots, sk_{i_n}^{(n)}$. The fact that all keys are contained in each encryption also disables hybrid arguments. To the best of our knowledge, the security of this protocol cannot be reduced to DKDM security.

Definition 5 (KDM security – tentative). *The oracle $\mathcal{T}$ chooses a random bit b and accepts the following queries.*

– *$pk(j)$ and $reveal(j)$: Return pk_j and sk_j, respectively. $dec(j,c)$: Return* $\mathrm{Dec}(sk_j, c)$.
– *$enc(j, f(i_1,\dots,i_t))$ where f is a circuit: Compute $m_0 := f(sk_{i_1},\dots,sk_{i_t})$ and return* $\mathrm{Enc}(pk_j, m_0)$.
– *$challenge(j, f(i_1,\dots,i_t))$: Compute m_0 as before, $m_1 := 0^{|m_0|}$ and return* $\mathrm{Enc}(pk_j, m_b)$.

The oracle aborts in the following cases: $reveal(j)$ is queried but $challenge(j,\cdot)$ has been queried before. $challenge(j,\cdot)$ is queried but $reveal(j)$ has been queried. $dec(j,c)$ is queried but c was produced by $challenge(j,\cdot)$. A scheme is KDM secure if no polynomial-time adversary guesses b with probability noticeably larger than $\frac{1}{2}$.

This definition might look appealing, but it cannot be met: For example, one could encrypt a challenge plaintext under pk_1 via the query $challenge(1,m)$, then encrypt the key sk_1 under pk_2 via $c := enc(2, sk_1)$, and finally reveal sk_2 via $reveal(2)$.[5] This sequence of queries is not forbidden by Definition 5. Now we can compute sk_1 from c using sk_2 and then decrypt the challenge encryption using sk_1. This allows to determine the bit b. Hence no encryption scheme can fulfill Definition 5. We hence have to relax the definition by excluding queries that would trivially allow to decrypt a challenge ciphertext. For this, we have to reject queries to the oracle that would allow the adversary to decrypt the challenge even in an ideal setting. For this, we keep track of the keys that the adversary can deduce from the queries made so far. We call this set *know* (the knowledge of the adversary) because it represents what the adversary knows *ideally.* The set *know* is inductively defined as follows: (a) If $reveal(j)$ has been queried, then $j \in know$. (b) If $j \in know$, and a $enc(j, f(i_1,\dots,i_t))$ has been queried, then $i_1,\dots,i_n \in know$. (c) If $enc(j, f(i_1,\dots,i_t))$ has been queried and returned the ciphertext c, and $dec(j,c)$ has subsequently been queried, then $i_1,\dots,i_t \in know$. Roughly, we say that the adversary knows all keys that either were revealed or are contained in ciphertexts it could decrypt using keys it knows. We can now relax Definition 5 by disallowing queries that would allow the adversary to know a secret key for a challenge encryption.

Definition 6 (KDM security – tentative). *KDM security is defined as in Definition 5 except that the oracle $\mathcal{T}$ additionally aborts if a query would lead to the following situation: For some $j \in know$, a query $challenge(j,\cdot)$ has been performed (or is being performed).*

Introducing hidden encryptions. Definition 6, however, is still to weak to allow to adaptively choose which keys to reveal. In particular, the example protocol given above can still not be proven secure: When producing c_1, c_2 in a reduction proof, we have to decide which of the ciphertexts will be created by challenge encryptions ($challenge(\cdot,\cdot)$ queries) and which will be created by normal encryptions ($enc(\cdot,\cdot)$). Since we might have to invoke $reveal(1)$ later, we may not use $challenge(1,\cdot)$ queries, and since we might have to invoke $reveal(2)$, we may not use $challenge(2,\cdot)$ queries. But if no $challenge(\cdot,\cdot)$ query is issued, the oracle $\mathcal{T}$ never uses the bit b and thus the adversary cannot guess b.[6]

[5] We use the shorthand m and sk_1 for the circuits outputting m and sk_1, respectively.

[6] Again, this problem might be remedied by guessing in advance whether sk_1 or sk_2 will be needed, but see footnote 4 for an example where guessing does not work.

Handling adaptive revelations of keys hence requires to further extend our approach. A closer inspection reveals why w we failed to prove the security of the example protocol: We had two possible ways to construct the ciphertext c_1. Either (a) we could ask the oracle to produce $c_1' := \mathrm{Enc}(pk_2, m\|sk_1\|sk_2)$ and encrypt it ourselves using pk_1 to produce c_1. Or (b) we could request the ciphertext c_1 directly by sending to the oracle a circuit f that computes c_1' from sk_1, sk_2. In case (a), we are not allowed to reveal sk_2 since this would allow to decrypt c_1' and thus reveal m. In case (b), if we were to reveal sk_1 this would allow to decrypt c_1. As the plaintext c_1' for c_1 has been produced using a circuit f from sk_1, sk_2 and m, the oracle has no way of knowing that c_1' is actually an encryption of these values (this would require an analysis of the circuit to determine what it does) and thus has to consider the values sk_1, sk_2 and m to be leaked when c_1 is decrypted. Thus in case (b), we have to disallow the revelation of sk_1. This analysis shows that we need a way to send the following instructions to the oracle: "First produce the ciphertext c_1' as an encryption of $m\|sk_1\|sk_2$ (where $m\|sk_1\|sk_2$ is described by a suitable circuit). Do *not* return the value c_1' (as otherwise we would be in case (a)). Then produce the ciphertext c_1 by encrypting c_1'. Return c_1."

Given these instructions, the oracle has enough information to deduce that when revealing sk_1, the message m is still protected by the encryption c_1' using pk_2 (the details of this deduction process are discussed below). And if only sk_2 is revealed instead, c_1 cannot be decryption and m is protected. Analogous reasoning applies to the construction of c_2.

Hence we have to define an oracle $\mathcal{T}$ that allows to construct ciphertexts without revealing them. Instead, for each ciphertext we can adaptively decide whether to reveal it or whether we only use it inside other ciphertexts (that again may or may not be revealed). More concretely, whenever a query is issued to $\mathcal{T}$, instead of directly returning the result of that query, it is stored in some register bit_h inside the oracle where h is a handle identifying the register. Only upon a special reveal query, the value bit_h is returned to the adversary. A challenge encryption (i.e., one whose content depends on the challenge bit b) is then produced as follows: First produce a plaintext m (possibly using a circuit and depending on other hidden strings) and assign it to register bit_{h_1}. Then, depending on b, assign bit_{h_1} or $0^{|bit_{h_1}|}$, respectively, to register bit_{h_2} (using a special challenge query $h_2 \leftarrow C(h_1)$). Encrypt bit_{h_2} using some key and assign the result to bit_{h_3}. Finally (optionally) reveal bit_{h_3}.[7]

These considerations lead to the following definition of the adKDM oracle (however, for the definition of adKDM security we will additionally define which sequences of queries are allowed):

Definition 7 (adKDM Oracle). *The* adKDM oracle $\mathcal{T}$ *maintains two partial functions cmd and bit (to increase readability we write bit*$_h$ *for bit*(h) *and cmd*$_h$ *for cmd*(h)*), a set* Φ*, a sequence of secret/public key pairs* sk_i, pk_i $(i \in \mathbb{N})$ *(which are generated when first accessed), and a bit* b *(the* challenge bit*). The function cmd will store the structure of previous queries, the function bit will store the corresponding bitstrings, and* Φ *will keep track of query results that are revealed to the adversary. We will refer to the elements in the domain of cmd and*

[7] This is, of course, not the only possible way to model challenge encryptions. One could, e.g., use a special command for producing a challenge encryption. However, we believe that the approach of being able to make challenge values out of arbitrary messages allows for more direct reductions in proofs. E.g., in our example protocol we could directly model the fact that m is the value that should remain hidden by using oracle call $h' \leftarrow C(h)$ when bit_h contains m and then using $bit_{h'}$ instead of bit_h in subsequent encryptions.

bit as handles in the following. Upon the first activation, b is chosen uniformly from $\{0,1\}$, bit and cmd are initially undefined, and Φ is empty. The oracle responds to the following commands:

– Encryption: $h' \leftarrow E(j,h)$ *where $cmd_{h'}$ has not been assigned, cmd_h has been assigned, and j is a key index: Set $bit_{h'} := \mathrm{Enc}(pk_j, bit_h)$ and $cmd_{h'} := E(j,h)$.*
– Decryption: $h' \leftarrow D(j,h)$ *where $cmd_{h'}$ has not been assigned, cmd_h has been assigned, and j is a key index: Set $bit_{h'} := \mathrm{Dec}(pk_j, bit_h)$, and $cmd_{h'} := D(j,h)$.*
– Circuit evaluation: $h' \leftarrow F(f, h_1, \dots, h_t)$ *where $cmd_{h'}$ has not been assigned, cmd_{h_i} has been assigned for all i, and f is a circuit with t arguments: Set $bit_{h'} := f(bit_{h_1}, \dots, bit_{h_t})$ and set $cmd_{h'} := F(f, h_1, \dots, h_t)$.*
– *Key request: $h' \leftarrow K(j)$ where $cmd_{h'}$ has not been assigned and j is a key index: Set $cmd_{h'} := K(j)$ and $bit_{h'} := sk_j$.*
– *Challenge: $h' \leftarrow C(h)$ where $cmd_{h'}$ has not been assigned and cmd_h has been assigned: Set $cmd_{h'} := C(h)$. If $b = 1$, set $bit_{h'} := bit_h$, otherwise set $bit_{h'} := 0^{|bit_h|}$.*
– *Reveal: $reveal(h)$ where cmd_h has been assigned: Add h to Φ and return bit_h.*
– Public key request: $pk(j)$ *where j is a key index: Return pk_j.*

The above commands in particular allow to assign a constant c to a handle h' by issuing $h' \leftarrow F(f)$ where f is a nullary circuit that returns c. We abbreviate this as $h' \leftarrow F(c)$. Note that the length of every bitstring is always known to the adversary, because Enc, Dec, and all f are length-regular.

The knowledge of the adversary. If $\mathcal{T}$ can be accessed in arbitrary ways, it is easy to determine b, e.g., querying $h_1 \leftarrow F(1)$, $h_2 \leftarrow C(h_1)$, $reveal(h_2)$ will return b. Thus we have to restrict the adversary to queries that will not trivially allow to deduce b. The necessary criteria are given below. In analogy to Definition 6 we do this by deriving a set *know* that characterizes what the adversary would ideally be able to know after the queries it performed. In contrast to Definition 6 the set *know* does not only contain keys, but the handles of all values produced by the oracle that the adversary would be able to know in an ideal setting. Intuitively, the knowledge *know* is defined by the following rules: All handles that the adversary requested (the set Φ) are considered known. If the decryption of a message is known, then that message is considered known.[8] If a circuit evaluation is known, all its arguments are considered known. If a challenge is known, the underlying message is considered known. If a key is known and an encryption of some message under that key is known, the message is considered known. And finally, if a decryption of some handle h_1 is known, and some handle h_2 evaluates to the same bitstring as h_1, and that handle h_2 resulted from an encryption of some message m, then that message m is considered known.

The last rule merits some additional explanation: The adversary may, e.g., construct and reveal an encryption c (assigned to some handle h_2) of some m. Then it constructs a circuit f that evaluates to c (by hard-coding c into f) and assigns $h_1 \leftarrow F(f)$. Now h_1 and h_2 refer to the same bitstring. By revealing the decryption of h_1, the adversary will then learn m. So after this sequence of queries, we have to ensure that m is considered known to the adversary.

[8] It may seem surprising that by learning the result of a decryption we may learn something about the ciphertext. However, in fact we can get a single bit about the ciphertext, namely whether it is valid or not. Combining this with the application of circuits, we can in principle retrieve the full ciphertext.

This is ensured by the last of the above rules. The following definition formally states the definition of the knowledge of the adversary.

Definition 8 (Knowledge). *For partial functions* cmd, bit *and a set* Φ*, we define the* knowledge $know = know_{cmd,bit,\Phi}$ of the adversary *to be inductively defined as follows:*

– $\Phi \subseteq know$.
– *If* $h' \in know$ *and* $cmd_{h'} = D(j, h)$ *then* $h \in know$.
– *If* $h' \in know$ *and* $cmd_{h'} = F(f, h_1, \ldots, h_t)$ *then* $h_1, \ldots, h_t \in know$.
– *If* $h' \in know$ *and* $cmd_{h'} = C(h)$ *then* $h \in know$.
– *If* $h' \in know$ *and* $cmd_{h'} = D(j, h_1)$, $bit_{h_1} = bit_{h_2}$ *and* $cmd_{h_2} = E(j, h_3)$ *then* $h_3 \in know$.
– *If* $h'_1, h'_2 \in know$ *and* $cmd_{h'_1} = K(j)$ *and* $cmd_{h'_2} = E(j, h)$ *then* $h \in know$.

Note that *know* can be efficiently computed given Φ, *cmd*, and *bit* by adding handles to *know* according to the rules in Definition 8 until *know* does not grow any more. We are now ready to state the final definition of adKDM security. Intuitively, an encryption scheme is adKDM secure if the probability that the adversary guesses b correctly without performing a query that would even ideally allow it to retrieve a bitstring constructed using a $C(\cdot)$ query.

Definition 9 (Adaptive KDM Security (adKDM)). *An encryption scheme* (Enc, Dec) *is* adKDM secure *if for any nonuniform polynomial-time adversary* A *there is a negligible function* μ *such that the following holds:*

$$\Pr[\mathsf{Guess} \wedge \neg\mathsf{Invalid}] \leq \tfrac{1}{2} + \mu(k)$$

where the events refer to an execution of A *with input* 1^k *and oracle access to* $\mathcal{T}_{(\mathrm{Enc},\mathrm{Dec})}$ *and the events are defined as follows:*

By Guess *we denote the event that the adversary outputs* b *where* b *is the challenge bit.*
By Invalid *we denote the event that* $h \in know_{cmd,bit,\Phi}$ *with* cmd_h *being of the form* $C(\cdot)$.

We will show that this definition can be met (at least in the random oracle model) in the next section. Clearly adKDM security implies DKDM security, since if we can only reveal keys that are not used for decrypting, the plaintexts of the challenge encryptions will never be in *know*.

On simulation-based notions. We often motivated our design choices above by comparison with an ideal setting in which the adversary knows exactly the bitstrings associated with handles in *know*. This leads to the question if it is be possible to instead directly define security under key-dependent message attacks using a simulation-based definition, i.e., to define an ideal functionality that handles encryption and decryption queries in an ideal fashion. This approach has been successfully used to formulate IND-CCA security in the UC framework [14]. Their approach, however, strongly depends on the fact that the functionality only needs to output public keys and (fake) encryptions (secret keys are only implicitly present due to the ability to use the functionality to decrypt messages).[9] It is currently unclear

[9] Technically, the reason is that a simulator has to be constructed that chooses the outputs of the functionality. As long as only public keys and ciphertexts are output, fake ciphertexts can be used since they cannot be decrypted. If the simulator had to generate secret keys, the fake ciphertexts could be decrypted and recognized.

how this approach could be extended to a functionality that can output secret keys. (It is of course possible to define a functionality that outputs secret keys as long as no encryption queries have been performed for that key, but this lead to a definition that is to weak to handle, e.g., our example protocol and that would roughly correspond to Definition 4.) This difficulty persists if we do not use the strong UC model [14] but instead the weaker stand-alone model as in [20, Chapter 7]. Consequently, although a simulation-based definition of KDM security might be very useful, it is currently unknown how to come up with such a definition.

4 OAEP is adKDM-Secure

We now prove the adKDM security of the OAEP scheme for a partial-domain one-way function. In particular, since the RSA permutation is partial-domain one-way under the RSA assumption [19], the adKDM security of RSA-OAEP follows.

Theorem 1 (OAEP is adKDM secure). *If f is a partial-domain one-way trapdoor 1-1 function, then the OAEP scheme* $(\mathrm{Enc}, \mathrm{Dec})$ *based on f is adKDM secure.*

To show this theorem, we first define an alternative characterization of partial-domain one-wayness.

Definition 10 (PD-Oracle). *The PD-oracle $\mathcal{P}_f$ for a trapdoor 1-1 function $f : S \times T \rightarrow$* range f *(that may depend on a security parameter) maintains sequences of public/secret key pairs sk_i, pk_i (generated on first use). It understands the following queries:*
- *$pk(j)$ and $sk(j)$: Return pk_j or sk_j, respectively.*
- *$challenge(h, j)$: If h has already been used, ignore this query. Let $j_h := j$. Choose (s_h, t_h) uniformly from $S \times T$. Set $c_h := f_{pk_{j_h}}(s_h, t_h)$. Return c_h.*
- *$decrypt(h)$: Return (s_h, t_h).*
- *$xdecrypt(c, j)$ where $(c, j) \neq (c_h, j_h)$ for all h. Check whether $f^{-1}_{sk_j}(c) = (s_h, t_h)$ for some h. If so, return (s_h, t_h). Otherwise return $\perp$.*
- *$check(s)$: Return the first h with $s_h = s$. If no such h exists, return $\perp$.*

By PDBreak *we denote the event that a query $check(s)$ is performed such that*
- *The query returns $h \neq \perp$.*
- *No query $sk(j_h)$ and no query $decrypt(h)$ has been performed before the current query.*

Lemma 1. *If f is partial-domain oneway, then for any nonuniform polynomial-time adversary A querying $\mathcal{P}_f$ we have that* $\Pr[\mathsf{PDBreak}]$ *is negligible in the security parameter.*

The proof is given in Appendix A.2. We additionally define a variant of the notion of knowledge as defined in Definition 8. We call this variant *lazy knowledge.*

Definition 11 (Lazy knowledge). *For partial functions cmd, bit and a set Φ, we define the* lazy knowledge *$lknow = lknow_{cmd,bit,\Phi}$ of the adversary to be inductively defined as follows:*
- *$\Phi \subseteq lknow$.*
- *If $h' \in lknow$ and $cmd_{h'} = D(j, h)$ then $h \in lknow$.*
- *If $h' \in lknow$ and $cmd_{h'} = F(f, h_1, \dots, h_t)$ then $h_1, \dots, h_t \in lknow$.*

– *If $h' \in lknow$ and $cmd_{h'} = C(h)$ then $h \in lknow$.*
– *If $h', h_1, h_2 \in lknow$, $cmd_{h'} = D(j, h_1)$, $bit_{h_1} = bit_{h_2}$ and $cmd_{h_2} = E(j, h_3)$ then $h_3 \in lknow$.*
– *If $h'_1, h'_2 \in lknow$ and $cmd_{h'_1} = K(j)$ and $cmd_{h'_2} = E(j, h)$ then $h \in lknow$.*

The only change with respect to Definition 8 is that in the fifth rule we require that $h_1, h_2 \in lknow$. In Definition 11 all rules depend only on values bit_h for which $h \in lknow$; thus one can efficiently compute $lknow$ without accessing bit_h for values $h \notin lknow$ by adding handles to $lknow$ according to these rules until $lknow$ does not grow any further. We call this algorithm the *lazy knowledge algorithm*. Note that $lknow \subseteq know$.

Proof sketch (of Theorem 1). To prove Theorem 1 we give a sequence of games that transforms an attack against the adKDM security of the OAEP scheme into an attack against the PD-oracle. This proof sketch only contains the proof structure and highlights selected steps.

GAME$_1$. The adversary A runs with access to the unmodified adKDM oracle $\mathcal{T}$. We assume that $\mathcal{T}$ invokes an encryption oracle $\mathcal{E}$ for encrypting and a decryption oracle $\mathcal{D}$ for decrypting. In particular, the encryption oracle $\mathcal{E}$ performs the following actions in the i-th query:

$$r \xleftarrow{\$} \{0,1\}^{k_0},\ g := G(r),\ s := (m \| 0^{k_1}) \oplus g,\ h := H(s),\ t := r \oplus h,\ c := f_{pk}(s,t).$$

The decryption oracle $\mathcal{D}$ acts as follows, assuming key index j and ciphertext c:
– $(s,t) := f^{-1}_{pk_j}(c)$, $r := t \oplus H(s)$, $(m, z) := s \oplus G(r)$ with $|m| = k - k_1 - k_0$ and $|z| = k_1$.
– If $z = 0^{k_1}$, return m, otherwise return $\perp$.

GAME$_4$.[10] We change the encryption oracle to first choose the ciphertext c and then compute the values s, t, r, h, t, g from it, i.e., upon the i-th query the encryption oracle does the following:

$$(s,t) \xleftarrow{\$} \{0,1\}^{k-k_0} \times \{0,1\}^{k_0},\ c \xleftarrow{\$} f_{pk}(s,t),\ r \xleftarrow{\$} \{0,1\}^{k_0},\ h := r \oplus t,\ g := (m \| 0^{k_1}) \oplus s$$

In particular, the values h and g are not retrieved from the oracles G and H any more. In order to keep the distribution of the values c, s, t, r, h, t, g consistent with the answers of the oracles G and H, the oracles G and H are additionally modified to return the values g and h chosen by the encryption oracle. We show that the probability of a successful attack is modified only by a negligible amount with respect to GAME$_1$.

GAME$_5$. We now change the definition of what constitutes a successful attack. In GAME$_1$–GAME$_4$, we considered it a successful attack if the adversary guessed the bit b chosen by the adKDM oracle $\mathcal{T}$ without performing queries such that the knowledge in the sense of Definition 8 would contain a handle corresponding to a query of the form $C(\cdot)$; see Definition 9.

Now, in GAME$_5$, we consider it to be a successful attack if the adversary guessed b without performing queries such that the *lazy* knowledge in the sense of Definition 11 does not contain a handle corresponding to a query $C(\cdot)$. Since the lazy knowledge is a subset of the knowledge, this represents a weakening of the restrictions put on the adversary. Thus the probability of an attack in GAME$_5$ is upper-bound by the probability of an attack in GAME$_4$.

[10] We keep the numbers of the games in sync with the full proof in the appendix.

Game_7. This step is arguably the most important step in the proof. In Game_5, bitstrings bit_h associated to handles h are often computed but never used. For example, the adversary might perform a query $h \leftarrow E(\dots)$ and never use the handle h again. More importantly, however, even if the adversary performs a query $h' \leftarrow E(j,h)$ for that handle h, the value bit_h does not need to be computed due to the following observation: The encryption oracle as introduced in Game_4 chooses the ciphertext c at random. The value g (which is the only value depending on the plaintext m) is only needed for suitably reprogramming the oracles G (namely such that $G(r) = g$). Thus we can delay the computation of g until G is queried at position r. Thus in case of a query $h' \leftarrow E(j,h)$, the value $m = bit_h$ is not needed for computing $bit_{h'}$. We use this fact to rewrite the whole game Game_5 such that it only computes a value bit_h when it is actually needed for computing some output sent to the adversary or for computing the lazy knowledge.

The bit b is only used in this game if a value bit_h is computed that corresponds to a query $h \leftarrow C(\cdot)$. If this is not the case, the communication between the adversary and $\mathcal{T}$ is independent of b. Hence, for proving that the probability of attack in the sense of Game_5 is only negligibly larger than $\frac{1}{2}$ (which then shows Theorem 1), it is sufficient to show that only with negligible probability, a value bit_h is computed such that h is not in the lazy knowledge. Namely, as long as no such value bit_h is computed, the adversary cannot have a higher probability in guessing b than $\frac{1}{2}$ unless $h \in \mathit{lknow}$.

Game_{10}. Now we replace the decryption oracle by a plaintext extractor. More concretely, the decryption oracle performs the following steps when given a ciphertext c:

(a) First, it checks whether $c = f_{pk}(s,t)$ for some pair (s,t) generated by the encryption oracle.[11] Then values (s,t) are known such that $f_{pk}(s,t) = c$, and the oracle can decrypt c without accessing the secret key sk.

(b) Otherwise, it checks whether for some s that has been computed by the encryption oracle, there exists a value t such that $f_{pk}(s,t) = c$. (Doing this efficiently requires the secret key; otherwise we had to iterate over all possible values t.) If so, reject the ciphertext.

(c) Otherwise, for all values s, r that have been generated so far, compute $t := r \oplus H(s)$ and $(m,z) = s \oplus G(r)$. Then check whether $f_{pk}(s,t) = c$ and $z = 0^{k_1}$. If so, return m. Otherwise reject the ciphertext.

We can show that this plaintext extractor is a good simulation of the original decryption oracle (in particular, the adversary is able to produce an s triggering rejection in (b) only if the decryption would fail anyway). Thus the probability that a value bit_h is computed such that h is not in the lazy knowledge does not increase by a non-negligible amount.

Game_{11}. In this final step, we modify Game_{10} not to generate the public/secret key pairs on its own, but to use the PD-oracle $\mathcal{P}$ defined in Definition 10. In particular, we make the following changes:

- When the secret key sk_j is needed (for computing bit_h for a $h \leftarrow K(j)$ query), query $sk(j)$ from $\mathcal{P}$.
- When producing a ciphertext $bit_{h'}$ (that are produced just to be random images of f_{pk}), use $\mathit{challenge}(h', j)$ where j is the corresponding key index.

[11] This does not imply that c has been generated by the encryption oracle since the encryption oracle might have used a different public key pk at that time.

- In the decryption oracle, for checking the condition (a) in GAME_{10}, we distinguish two cases. If c was produced by the encryption oracle the decryption oracle sends a *decrypt*(h) to $\mathcal{P}$ where h is the query where c was produced. Otherwise it sends an *xdecrypt*(c, j) query to $\mathcal{P}$ where j is the index of the key used in the decryption query. In both cases, if the check in (a) would have succeeded, $\mathcal{P}$ will send back a preimage (s, t) of c.
- The check (b) is performed by sending *check*(s) to $\mathcal{P}$.

A case analysis reveals that if a value bit_h is computed such that h is not in the lazy knowledge, then the event $\mathsf{PDBreak}$ (as in Definition 10) occurs. By Lemma 1 this can only happen with negligible probability. Thus no value bit_h is computed such that h is not in the lazy knowledge, and therefore the advantage of the adversary is negligible (as discussed in GAME_7). □

A Postponed proofs

In this appendix, we give the proofs of Theorem 1 and Lemma 1. These proofs will not appear in the proceedings version of this paper. They are included for the referee's convenience.

A.1 Proof of Theorem 1

Convention: Encryption is length-regular. For any encryption scheme, we impose the following assumption on the output of the encryption function Enc and the decryption function Dec: The length of the output of Enc depends only on the public key and the length of the message. The length of the output of Dec depends only on the *public* key and the length of the ciphertext. In particular, if Dec can output a special symbol $\perp$ denoting a failed encryption/decryption, this symbol has to be encoded to have the same length as a valid decryption. This can be assumed without loss of generality since Enc has to be length-regular anyway (otherwise the encryption scheme would not even be IND-CPA secure), and since, given an upper bound l on the length of the plaintext, we can, e.g., let Dec pad its output with $1\|0^*$ to length $l+1$. Our definition below does not require a decryption of an encryption to yield the plaintext, and that the adversary may insert arbitrary (length-regular) conversion functions to reverse that encoding. In the case of OAEP encryption, we will assume for concreteness that the encryption Enc first applies the function *pad* to its argument which truncates or pads its argument to length $k-k_0-k_1$ with 0^*, and the decryption Dec applies the function *encode* to its output that encodes m as $1\|m$ and $\perp$ as $0^{k-k_0-k_1+1}$.

Notation. Let A be a polynomial-time adversary. Fix a security parameter k. Assume that A performs at most q queries (to $\mathcal{T}$, G, and H together). We use the following notation: By $\Pr_i[X]$ we denote the probability of event X in $\textsf{Game}_i$. $\mathcal{T}$ is the adKDM oracle. We assume that $\mathcal{T}$ uses oracles $\mathcal{E}$ and $\mathcal{D}$ for encryption and decryption. $\mathcal{E}$ and $\mathcal{D}$ access random oracles G and H for the OAEP construction. Further, $\mathcal{T}$ may also access G and H if evaluating a circuit requires it. Of course, the adversary also has access to G and H. The parameters k_0 and k_1 are as in the definition of OAEP. We enumerate all queries made by the adversary using a common index $i=1,\dots,q$, including queries to G and H and to $\mathcal{T}$. By $m_h, g_h, h_h, s_h, r_h, t_h, c_h$ we denote the corresponding values from the definition of the encryption of OAEP used in the query $h \leftarrow E(j,h')$ to $\mathcal{T}$ (the variables are undefined if no such query is made), and j_h denotes the key index j used in that query. By q_E and q_D we denote the number of encryption and decryption queries performed by the adversary. By q_G and q_H we denote the number of queries to G and H performed by the adversary and by circuits evaluated by $\mathcal{T}$ (but not queries made by $\mathcal{E}$ or $\mathcal{D}$). Note that the encryption pads its input using the function *pad* and the decryption encodes its output using the function *encode* (see the discussion after Definition 1). For readability, we omit these function applications, the reader should keep in mind that values m and m_h (denoting plaintexts) are implicitly subjected to these transformations.

Proof (of Theorem 1). To prove Theorem 1 we will give a sequence of games that transforms an attack against the adKDM security of the OAEP scheme into an attack against the PD-oracle. In each game, we highlight the changed parts using a bold font.

$\textsf{Game}_1$. The adversary A runs with access to the unmodified adKDM oracle $\mathcal{T}$. In particular, the encryption oracle performs the following actions in a query $h \leftarrow E(j,h)$:

$$r_h \xleftarrow{\$} \{0,1\}^{k_0},\ g_h := G(r_h),\ s_h := (m_h \| 0^{k_1}) \oplus g_h,\ h_h := H(s_h),\ t_h := r_h \oplus h_h,\ c_h := f_{pk_j}(s_h, t_h).$$

The decryption oracle does, assuming key index j and ciphertext c, the following:
– $(s,t) := f^{-1}_{pk_j}(c)$, $r := t \oplus H(s)$, $(m,z) := s \oplus G(r)$ with $|m| = k - k_1 - k_0$ and $|z| = k_1$.
– If $z = 0$, return m, otherwise return $\perp$.

$\textsf{Game}_2$. We now modify the encryption oracle not to use the oracle G any more but to choose g_h uniformly at random. That is, the encryption oracle performs the following steps in a query $h \leftarrow E(j,h')$ with $m_h := bit_{h'}$:

$$r_h \xleftarrow{\$} \{0,1\}^{k_0},\ \boldsymbol{g_h \xleftarrow{\$} \{0,1\}^{k-k_0}},\ s_h := (m_h \| 0^{k_1}) \oplus g_h,$$
$$h_h := H(s_h),\ t_h := r_h \oplus h_h,\ c_h := f_{pk_j}(s_h, t_h).$$

Furthermore, we modify the oracle G to match the choice of g_h, that is, the oracle G is modified as follows:
– If $r = r_h$ for some h, return g_h.
– Otherwise, return a lazily sampled value (i.e., if r has been queried before, return the value returned before, otherwise, return fresh randomness).

It is easy to see that $\textsf{Game}_1$ and $\textsf{Game}_2$ differ only if some r_h is queried from G before the encryption query $h \leftarrow E(\cdot,\cdot)$ has been performed (i.e., if G is queried at a position that will later be assigned another value). Since r_h will be randomly chosen in the encryption query, the probability that in a given G-query we query $G(r_h)$ is bounded by $1/2^{k-k_0}$. Since there are at most $q_G + q_D$ such G-queries, we have that

$$\left|\Pr_1[\mathsf{Guess} \wedge \neg\mathsf{Invalid}] - \Pr_2[\mathsf{Guess} \wedge \neg\mathsf{Invalid}]\right| \leq \frac{q_G + q_D}{2^{k-k_0}}. \tag{1}$$

$\textsf{Game}_3$. We now modify the encryption oracle not to use the oracle H any more but to choose h_i uniformly at random, i.e., the encryption oracle performs the following steps in a query $h \leftarrow E(j,h')$:

$$r_h \xleftarrow{\$} \{0,1\}^{k_0},\ g_h \xleftarrow{\$} \{0,1\}^{k-k_0},\ s_h := (m_h \| 0^{k_1}) \oplus g_h,$$
$$\boldsymbol{h_h \xleftarrow{\$} \{0,1\}^{k_0}},\ t_h := r_h \oplus h_h,\ c_h := f_{pk_{j_h}}(s_h, t_h).$$

Furthermore, we modify the oracle H to match the choice of h_h, i.e., the oracle H is modified as follows:
– If $s = s_h$ for some h, return h_h.
– Otherwise, return a lazily sampled value.

Again, it is easy to see that $\textsf{Game}_2$ and $\textsf{Game}_3$ differ only if some s_h is queried from H before the encryption query $h \leftarrow E(\cdot,\cdot)$. Since $s_h = (m_h \| 0^{k_1}) \oplus g_h$ and g_h will be randomly chosen in the encryption query, the probability that in that H-query we query $H(s_h)$ is bounded by $1/2^{k_1}$. Since there are at most $q_H + q_D$ such H-queries, we have that

$$\left|\Pr_2[\mathsf{Guess} \wedge \neg\mathsf{Invalid}] - \Pr_3[\mathsf{Guess} \wedge \neg\mathsf{Invalid}]\right| \leq \frac{q_H + q_D}{2^{k_1}}. \tag{2}$$

$\textsf{Game}_4$. We change the encryption oracle to first choose the ciphertext c_h and then compute the values $s_h, t_h, r_h, h_h, t_h, g_h$ from it, i.e., upon the i-th query the encryption oracle does the following:

$$\boldsymbol{(s_h, t_h) \xleftarrow{\$} \{0,1\}^{k-k_0} \times \{0,1\}^{k_0}},\ c_h \xleftarrow{\$} f_{pk_{j_h}}(s_h, t_h),\ r_h \xleftarrow{\$} \{0,1\}^{k_0},$$
$$\boldsymbol{h_h := r_h \oplus t_h},\ g_h := (m_h \| 0^{k_1}) \oplus s_h$$

This does not change the distribution of any of $c_h, s_h, t_h, r_h, h_h, t_h, g_h$. Thus we have

$$\Pr_3[\textsf{Guess} \wedge \neg\textsf{Invalid}] = \Pr_4[\textsf{Guess} \wedge \neg\textsf{Invalid}]. \tag{3}$$

$\textsf{Game}_5$. We modify $\textsf{Game}_4$ by adding the following: For each i, after the i-th query, $\mathcal{T}$ computes the lazy knowledge using the lazy knowledge algorithm and assigns that lazy knowledge (i.e., a set of handles) to $\textsf{LKnow}_i$. Denote by $\textsf{Invalid2}$ the event that for some i and some handle h, we $h \in \textsf{LKnow}_i$ and $cmd_h = C(\cdot)$. Let *lknow* denote the lazy knowledge and *know* the knowledge after the execution of $\textsf{Game}_5$. Then $\textsf{LKnow}_i \subseteq \mathit{lknow} \subseteq \mathit{know}$, thus $\textsf{Invalid2}$ implies $\textsf{Invalid}$. Since the output of the lazy knowledge algorithm is not used and the algorithm has no side effects, we have

$$\Pr_4[\textsf{Guess} \wedge \neg\textsf{Invalid}] = \Pr_5[\textsf{Guess} \wedge \neg\textsf{Invalid}] \le \Pr_5[\textsf{Guess} \wedge \neg\textsf{Invalid2}]. \tag{4}$$

$\textsf{Game}_6$. Let $\mathfrak{H}$ be some set of handles (that may depends on the state of the oracle $\mathcal{T}$ at the beginning of the query). We will specify $\mathfrak{H}$ only when we define $\textsf{Game}_7$ below. We now let the decryption oracle $\mathcal{D}$ check whether H is queried with a value s such that $s = s_h$ for some $h \in \mathfrak{H}$. In this case we reject the query. However, we perform this only for ciphertexts that do not have the same preimage as some other ciphertext generated by $\mathcal{T}$.[12] We perform this check before querying G or H. That is, the decryption oracle performs the following actions upon a query with key index j and ciphertext c:

– Compute $(s,t) := f^{-1}_{pk_j}(bit_h)$.
– Check whether $(s,t) = (s_{\tilde{h}}, t_{\tilde{h}})$ for some $\tilde{h} \in \mathfrak{H}$. If so, compute

$$r := t \oplus H(s),\ (m,z) := s \oplus G(r) \text{ with } |m| = k - k_1 - k_0 \text{ and } |z| = k_1.$$

If $z = 0^{k_1}$, return m. Otherwise return $\perp$.
– Otherwise, perform the following steps:
  (i) **Check whether $\boldsymbol{s = s_{\tilde{h}}}$ for some $\boldsymbol{\tilde{h} \in \mathfrak{H}}$. If so, return $\boldsymbol{\perp}$.**
  (ii) Compute

$$r := t \oplus H(s),\ (m,z) := s \oplus G(r) \text{ with } |m| = k - k_1 - k_0 \text{ and } |z| = k_1.$$

  (iii) Check whether $z = 0^{k_1}$. If not, return $\perp$.
  (iv) Return m.

[12] This is not the same as saying that the ciphertexts are equal since both ciphertexts may have been generated using different keys.

Let E denote the event that in some query to $\mathcal{D}$, we have that $(s,t) \neq (s_{\tilde{h}}, t_{\tilde{h}})$ for all $\tilde{h} \in \mathfrak{H}$, $s = s_{\tilde{h}}$ for some $\tilde{h} \in \mathfrak{H}$ and $z = 0^{k_1}$. Since this is the only case in which the new check (i) returns $\perp$ although the check (iii) would not return $\perp$, we have that $\left|\Pr_5[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}] - \Pr_6[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}]\right| \leq \Pr_6[E]$. We therefore have to bound $\Pr_6[E]$. Assume that E occurs. For some $x \in \{0,1\}^{k-k_0}$, let x' denote the last two bits of x. Since $z = 0^{k_1}$, we have that $G(r_{\tilde{h}})' = 0^{k_1} \oplus s'_{\tilde{h}} = z \oplus s' = G(r)'$. Furthermore, since $t \neq t_{\tilde{h}}$, we have $r = t + H(s) = t + H(s_{\tilde{h}}) \neq t_{\tilde{h}} + H(s_{\tilde{h}}) = r_{\tilde{h}}$. Thus E only occurs if $G(r_{\tilde{h}})' = G(r)'$ but $r_{\tilde{h}} \neq r$. Since the values of G are independently uniformly chosen for different r, the probability that the last k_1 bits of two of the images collide is bounded by $t^2/2^{k_1}$ where t is the number of images that are sampled for G. Since G is queried at most $q_D + q_G$ times and in each encryption query at most one value $g_{\tilde{h}}$ is fixed for G (see GAME_2), we have $t \leq q_D + q_G + q_E$. Therefore we have that

$$\left|\Pr_5[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}] - \Pr_6[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}]\right| \leq \Pr_6[E] \leq \frac{(q_D + q_G + q_E)^2}{2^{k_1}}. \tag{5}$$

GAME_7. We now change GAME_6 such that the values $c_h, s_h, t_h, r_h, h_h, t_h, g_h, m_h$, as well as the bitstrings bit_h are computed lazily (on-demand). More exactly, $\mathcal{T}$ is changed to behave as follows:

- Upon a query X where X is one of $h' \leftarrow E(j,h)$, $h' \leftarrow D(j,h)$, $h' \leftarrow F(f, h_1, \dots, h_t)$, $h' \leftarrow K(j)$, $h' \leftarrow C(h)$, add X to the trace but *do not compute* the corresponding bitstring $bit_{h'}$.
- Upon a query $reveal(h)$, request the value bit_h (see below), and return bit_h.
- Upon a query $pk(j)$, return pk_j.
- If G is queried with some value r, check whether $r = r_h$ for some h such that r_h has already been set. If so, return g_h. Otherwise, return a lazily sampled value $G(r)$.
- If H is queried with some value s, check whether $s = s_{h'}$ for some h' such that $s_{h'}$ has already been set (note that in this case, $c_{h'}$ and $t_{h'}$ have also been set). If so, we have $cmd_{h'} = E(j,h)$. Let h be the handle occurring in $cmd_{h'} = E(j,h)$, and request bit_h and compute

  $$m_{h'} := bit_h,\ r_{h'} \xleftarrow{\$} \{0,1\}^{k_0},\ h_{h'} := r_{h'} \oplus t_{h'},\ g_{h'} := (m_{h'} \| 0^{k_1}) \oplus s_{h'}.$$

  Then return $h_{h'}$. On the other hand, if no such h' exists, return a lazily sampled value $H(s)$.
- When the value $bit_{h'}$ is requested that has not been requested before, do the following:
  - If $cmd_{h'} = E(j,h)$, choose $(s_h, t_h) \xleftarrow{\$} \{0,1\}^{k-k_0} \times \{0,1\}^{k_0}$ and set $bit_{h'} := c_h := f_{pk_j}(s_h, t_h)$ and return $bit_{h'}$.
  - If $cmd_{h'} = D(j,h)$, request bit_h and invoke the decryption oracle $\mathcal{D}$ with inputs j and $c := bit_h$.
  - If $cmd_{h'} = F(f, h_1, \dots, h_t)$, request $bit_{h_1}, \dots, bit_{h_t}$ and set $bit_{h'} := f(bit_{h_1}, \dots, bit_{h_t})$.
  - If $cmd_{h'} = K(j)$, set $bit_{h'} := sk_j$.
  - If $cmd_{h'} = C(h)$, request bit_h and let $bit_{h'} := bit_h$ if $b = 1$ and $bit_{h'} := 0^{|bit_h|}$ if $b = 0$.
- After the i-th query, $\mathcal{T}$ computes LKnow_i using the lazy knowledge algorithm (and requesting bit_h only when the lazy knowledge algorithm needs to access bit_h).

Furthermore, now specify the set of handles $\mathfrak{H}$ introduced in Game_6. In a query to the decryption oracle, we let $\mathfrak{H}$ be those handles $\tilde{h}$ such that $s_{\tilde{h}}$ and $t_{\tilde{h}}$ have already been defined (i.e., have already been needed for the lazy evaluation). Note that $\mathfrak{H}$ is also well-defined in Game_6, since in Game_6, one can simulate the whether $s_{\tilde{h}}$ or $t_{\tilde{h}}$ would already have been set in Game_7 and thus whether $\tilde{h}$ should be in $\mathfrak{H}$.

With this choice of $\mathfrak{H}$, $\mathcal{D}$ is defined as in Game_6, except that we can now omit the condition $\tilde{h} \in \mathfrak{H}$ for $\tilde{h}$ since now only those $s_{\tilde{h}}$ are defined that satisfy $\tilde{h} \in \mathfrak{H}$, so $\tilde{h} \in \mathfrak{H}$ is automatically satisfied.

Let b denote the challenge bit chosen by $\mathcal{T}$. Let b' denote the output of the adversary. Let $\mathsf{PredictG}$ denote the event that at some point, a query $G(r)$ is executed and that *later* some r_h will be assigned a value r' with $r = r'$. Let $\mathsf{PredictH}$ denote the event that at some point, a query $H(s)$ is executed and that *later* some s_h will be assigned a value s' with $s = s'$.

Fix a random tape for the adversary, and fix an outcome for each random choice performed in the games Game_6 and Game_7 (e.g., choice of pk_i, choice of r_h, etc.)

Then Game_7 computes the same values $\mathsf{LKnow}_1, \dots, \mathsf{LKnow}_q$, b, b' as does Game_6 unless $\mathsf{PredictG}$ or $\mathsf{PredictH}$ occur in Game_7. Thus in particular, over all random choices, $\mathsf{LKnow}_1, \dots, \mathsf{LKnow}_q$, b, b' have the same distribution in Game_6 and Game_7 unless $\mathsf{PredictG}$ or $\mathsf{PredictH}$ occur in Game_7. Since the event $\mathsf{Invalid2}$ is defined only in terms of LKnow_i, and since the event Guess is defined only in terms of b and b', it follows that

$$\left|\Pr_6[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}] - \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}]\right| \le \Pr_7[\mathsf{PredictG}] + \Pr_7[\mathsf{PredictH}].$$

We now bound $\Pr_7[\mathsf{PredictG}]$ and $\Pr_7[\mathsf{PredictH}]$. The event $\mathsf{PredictG}$ occurs if a query $G(r)$ is performed such that later some r_h will be assigned the value r. Since the r_h will then be chosen uniformly with $|r_h| = k_0$, and since there are at most $q_G + q_D$ queries of G and at most q_E different r_h will be assigned, we have $\Pr_7[\mathsf{PredictG}] \le \frac{q_G+q_D}{2^{k_0}}$. Similarly, we have that $\Pr_7[\mathsf{PredictH}] \le \frac{q_H+q_D}{2^{k-k_0}}$. We hence altogether obtain

$$\left|\Pr_6[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}] - \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}]\right| \le \frac{q_G + q_D}{2^{k_0}} + \frac{q_H + q_D}{2^{k-k_0}}. \qquad (6)$$

By $\mathsf{ComputeUnknown}$ we denote the event that in the i-query, bit_h is requested for some $h \notin \mathsf{LKnow}_i$.

If $\neg\mathsf{Invalid2}$, then there will be no h, i with $h \in \mathsf{LKnow}_i$ such that cmd_h is of the form $C(\cdot)$. Thus $\neg\mathsf{ComputeUnknown} \wedge \neg\mathsf{Invalid2}$ implies that no bit_h will be requested where cmd_h is of the form $C(\cdot)$. Since in Game_7 the challenge bit b is only used when bit_h is requested for some h with cmd_h of the form $C(\cdot)$, it follows that b will only be used if $\mathsf{ComputeUnknown} \vee \mathsf{Invalid2}$. Let BUsed denote the event that the challenge bit is used. We have $\neg\mathsf{ComputeUnknown} \vee \neg\mathsf{Invalid2} \Rightarrow \neg\mathsf{BUsed}$

and $\Pr_7[\mathsf{Guess}|\neg\mathsf{BUsed}] = \frac{1}{2}$, thus

$$\begin{aligned}
\Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2}] &= \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2} \wedge \neg\mathsf{BUsed}] \\
&\quad + \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2} \wedge \mathsf{ComputeUnknown} \wedge \mathsf{BUsed}] \\
&\quad \left.+ \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2} \wedge \neg\mathsf{ComputeUnknown} \wedge \mathsf{BUsed}] \quad \right\} = 0 \\
&= \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2} \mid \neg\mathsf{BUsed}] \cdot \Pr_7[\neg\mathsf{BUsed}] \\
&\quad + \Pr_7[\mathsf{Guess} \wedge \neg\mathsf{Invalid2} \wedge \mathsf{ComputeUnknown} \wedge \mathsf{BUsed}] \\
&\le \tfrac{1}{2} \Pr_7[\neg\mathsf{BUsed}] + \Pr_7[\mathsf{ComputeUnknown}] \\
&\le \tfrac{1}{2} + \Pr_7[\mathsf{ComputeUnknown}]. \qquad (7)
\end{aligned}$$

GAME_8. We now add another check to the decryption oracle $\mathcal{D}$: We check whether the value r occurring during decryption has been queried from G before (by someone other than $\mathcal{D}$). If not, the ciphertext is rejected. However, we do this only for ciphertexts that do not have the same preimage as some other ciphertext generated by $\mathcal{T}$. More exactly, the decryption oracle performs the following actions upon a query with key index j and ciphertext c:

- Compute $(s,t) := f^{-1}_{pk_j}(bit_h)$.
- Check whether $(s,t) = (s_{\tilde h}, t_{\tilde h})$ for some handle $\tilde h$. If so, compute

$$r := t \oplus H(s),\ (m,z) := s \oplus G(r) \text{ with } |m| = k - k_1 - k_0 \text{ and } |z| = k_1.$$

  If $z = 0^{k_1}$, return m. Otherwise return $\perp$.
- Otherwise, perform the following steps:
  (i) Check whether $s = s_{\tilde h}$ for some handle $\tilde h$. If so, return $\perp$.
  (ii) Compute $r := t \oplus H(s)$.
  (iii) **Check whether $r = r_{\tilde h}$ for some $\tilde h$ or $G(r)$ has been queried before. If not, return $\perp$. Otherwise, let g^* denote $g_{\tilde h}$ or the value that was returned by the last query of $G(r)$, respectively.**
  (iv) Compute $(m,z) := s \oplus g^*$ with $|m| = k - k_1 - k_0$ and $|z| = k_1$.
  (v) Check whether $z = 0^{k_1}$. If not, return $\perp$.
  (vi) Return m.

To bound $\left|\Pr_7[\mathsf{ComputeUnknown}] - \Pr_8[\mathsf{ComputeUnknown}]\right|$ we first introduce a hybrid game GAME_8^m for $i = 0,\dots,q$. The first m queries, this game acts like GAME_8 (i.e., the check (iii) is performed). In the remaining queries, the game acts like GAME_7 (i.e., the check (iii) is not performed). Obviously, GAME_8^0 is equivalent to GAME_7 and GAME_8^q to GAME_8. Let $\Pr^m[X]$ denote the probability of an event X in GAME_8^m. Fix some $m = 1,\dots,q$ and compare GAME_8^{m-1} and GAME_8^m. To bound $\left|\Pr^m[\mathsf{ComputeUnknown}] - \Pr^{m-1}[\mathsf{ComputeUnknown}]\right|$ by some δ, we have to see two things. First, the probability of the following event is bounded by δ: In the m-th query we have $r \neq r_{\tilde h}$ for all $\tilde h$ and $G(r)$ has never been queried before but the check whether $z = 0^{k_1}$ would succeed. Second, in case we return $\perp$ in (iii), $G(r)$ is not queried in (iv). So we have to see that the remainder of the execution is independent of any side effects of querying G. The first follows from the fact that if $r \neq r_{\tilde h}$ for all $\tilde h$ and $G(r)$ has never been queried before, then $G(r)$ will return a fresh random value, so the probability that $z = 0^{k_1}$ is bounded by $\delta := 2^{-k_1}$. To show the second fact, note that querying G only has a side effect if some part of $\mathcal{T}$ depends on the list of queries performed so far. Since in the hybrid game Game_8^{m+1} only the check (iii) depends on that list, and this check is not executed after the m-th query, omitting the query to G in the m-th query has no effect on the later execution. Thus we have

$$\left|\Pr_7[\mathsf{ComputeUnknown}] - \Pr_8[\mathsf{ComputeUnknown}]\right| \le \frac{q}{2^{k_1}}. \qquad (8)$$

Game_9. We now add another check to the decryption oracle $\mathcal{D}$, namely whether the value s occurring during decryption has been queried from H before (by someone other than $\mathcal{D}$). If not, the ciphertext is rejected. Again, we do this only for ciphertexts that do not have the same preimage as some other ciphertext generated by $\mathcal{T}$. More exactly, the decryption oracle performs the following actions upon a query with key index j and ciphertext c:

– Compute $(s,t) := f_{pk_j}^{-1}(bit_h)$.

– Check whether $(s,t) = (s_{\tilde{h}}, t_{\tilde{h}})$ for some handle $\tilde{h}$. If so, compute

$$r := t \oplus H(s),\ (m,z) := s \oplus G(r) \text{ with } |m| = k - k_1 - k_0 \text{ and } |z| = k_1.$$

If $z = 0^{k_1}$, return m. Otherwise return $\perp$.

– Otherwise, perform the following steps:
   (i) Check whether $s = s_{\tilde{h}}$ for some handle $\tilde{h}$. If so, return $\perp$.
   (ii) **Check whether $\boldsymbol{H(s)}$ has been queried before. If not, return $\boldsymbol{\perp}$. Otherwise, let $\boldsymbol{h^*}$ denote the value that was returned by the last query of $\boldsymbol{H(s)}$.**
   (iii) Compute $r := t \oplus h^*$.
   (iv) Check whether $r = r_{\tilde{h}}$ for some $\tilde{h}$ or $G(r)$ has been queried before. If not, return $\perp$. Otherwise, let g^* denote $g_{\tilde{h}}$ or the value that was returned by the last query of $G(r)$, respectively.
   (v) Compute $(m,z) := s \oplus g^*$ with $|m| = k - k_1 - k_0$ and $|z| = k_1$.
   (vi) Check whether $z = 0^{k_1}$. If not, return $\perp$.
   (vii) Return m.

Note that a query of $H(s)$ only leads to requesting bit_h if $s = s_{\tilde{h}}$ for some $\tilde{h}$ (see the description of Game_7). However, we exclude this condition by the check (i). Further, note that if $H(s)$ is a fresh random value, then r is also a fresh random value. In this case the probability that r has been queried from G or that $r = r_{\tilde{h}}$ for some $\tilde{h}$ is bounded by $(q_G + q_D + q_E)/2^{k_0}$ (since there are at most $q_G + q_D$ queries to G and at most q_E values $r_{\tilde{h}}$ are defined and $|r| = k_0$). Thus, analogously as for (8) we get

$$\left|\Pr_8[\mathsf{ComputeUnknown}] - \Pr_9[\mathsf{ComputeUnknown}]\right| \le \frac{q \cdot (q_G + q_D + q_E)}{2^{k_0}}. \qquad (9)$$

Game_{10}. We now modify $\mathcal{D}$ as follows: For ciphertexts c that do not have the same preimage as some other ciphertext generated by $\mathcal{T}$, we loop over all s,t that have been queried from G and H, and check whether $c = f_{pk_j}(s,t)$. This will later allow to remove the invocation of $f_{pk_j}^{-1}(c)$. More exactly, the decryption oracle performs the following actions upon a query with key index j and ciphertext c:

– Check whether $f_{pk_j}^{-1}(c) = (s_{\tilde{h}}, t_{\tilde{h}})$ for some handle $\tilde{h}$. If so, compute

$$r := t_{\tilde{h}} \oplus H(s_{\tilde{h}}),\ (m,z) := s_{\tilde{h}} \oplus G(r) \text{ with } |m| = k - k_1 - k_0 \text{ and } |z| = k_1.$$

If $z = 0^{k_1}$, return m. If $z \neq 0^{k_1}$, return $\bot$.

- Otherwise, perform the following steps:
  - Check whether $s = s_{\tilde{h}}$ for some handle $\tilde{h}$. If so, return $\bot$.
  - Let Q_r be the set of all r that have been queried from G before by someone other than $\mathcal{D}$ and of all r such that $r = r_{\tilde{h}}$ for some handle $\tilde{h}$.
  - Let Q_s be the set of all s that have been queried from H before by someone other than $\mathcal{D}$.
  - For all $(r, s) \in Q_r \times Q_s$, perform the following steps:
    * Let g^* denote $g_{\tilde{h}}$ or the value returned by the last query $G(r)$, respectively, and let and h^* denote the value returned by the last query $H(s)$.
    * Compute $t := r \oplus h^*$ and check whether $f_{pk_j}(s,t) = c$. If not, continue with the next pair (r, s).
    * Compute $(m, z) := s \oplus g^*$ with $|m| = k - k_1 - k_0$ and $|z| = k_1$.
    * If $z = 0^{k_1}$, return m. Otherwise return $\bot$.
  - If none of the (r, s) satisfied all conditions in the loop (i.e., if the loop is left without incurring a return statement), return $\bot$.

It is easy to see that this oracle $\mathcal{D}$ returns $\bot$ if and only if the oracle $\mathcal{D}$ from GAME_9 returns $\bot$. Further, if this oracle $\mathcal{D}$ returns some $m \neq \bot$, then it is the same m that the oracle $\mathcal{D}$ from GAME_9 would have returned, since the values r, s, t, m, z are uniquely determined by c (although it may be computationally hard to find them).

Since the output of $\mathcal{D}$ did not change, and since in both games $\mathcal{D}$ has no side effects for ciphertexts that do not have the same preimage as some other ciphertext generated by $\mathcal{T}$, we have

$$\Pr_9[\mathsf{ComputeUnknown}] = \Pr_{10}[\mathsf{ComputeUnknown}]. \tag{10}$$

GAME_{11}. We now modify the GAME_{10} not to generate the private and public keys by itself, and not to apply f_{pk} and f_{sk}^{-1} by itself, but to invoke an instance of the PD-oracle $\mathcal{P}$ instead (Definition 10). More exactly, the oracle $\mathcal{T}$ now performs as follows:

- Upon a query X where X is one of $h' \leftarrow E(j,h)$, $h' \leftarrow D(j,h)$, $h' \leftarrow F(f, h_1, \ldots, h_t)$, $h' \leftarrow K(j)$, $h' \leftarrow C(h)$, add X to the trace but *do not compute* the corresponding bitstring $bit_{h'}$.
- Upon a query $reveal(h)$, request the value bit_h (see below), and return bit_h.
- Upon a query $pk(j)$, **query $\boldsymbol{pk(j)}$ from $\boldsymbol{\mathcal{P}}$, let $\boldsymbol{pk_j}$ be the response,** and return pk_j.
- If G is queried with some value r, check whether $r = r_h$ for some h such that r_h has already been set. If so, return g_h. Otherwise, return a lazily sampled value $G(r)$.
- If H is queried with some value s, **query $\boldsymbol{check(s)}$ from $\boldsymbol{\mathcal{P}}$ and let the answer be $\boldsymbol{h'}$. If $\boldsymbol{h' \neq \bot}$, retrieve $\boldsymbol{(s_{h'}, t_{h'})}$ from $\boldsymbol{\mathcal{P}}$ using query $\boldsymbol{decrypt(h')}$.** Since $s_{h'}$ is defined, $cmd_{h'} = E(j,h)$. Let h be the handle occurring in $cmd_{h'} = E(j,h)$. Then request bit_h and compute

$$m_{h'} := bit_h,\ r_{h'} \xleftarrow{\$} \{0,1\}^{k_0},\ h_{h'} := r_{h'} \oplus t_{h'},\ g_{h'} := (m_{h'} \| 0^{k_1}) \oplus s_{h'}.$$

  Then return $h_{h'}$. On the other hand, **if $\boldsymbol{h' = \bot}$**, return a lazily sampled value $H(s)$.
- When the value $bit_{h'}$ is requested that has not been requested before, do the following:

- If $cmd_{h'} = E(j, h)$, **query $c_{h'}$ from $\mathcal{P}$ using the query *challenge*(h', j) and return $c_{h'}$.**
- If $cmd_{h'} = D(j, h)$, first request bit_h. and invoke the decryption oracle $\mathcal{D}$ with inputs j and $c := bit_h$.
- If $cmd_{h'} = F(f, h_1, \dots, h_t)$, request $bit_{h_1}, \dots, bit_{h_t}$ and set $bit_{h'} := f(bit_{h_1}, \dots, bit_{h_t})$.
- If $cmd_{h'} = K(j)$, **request sk_j from $\mathcal{P}$ using the query *sk*(j)** and set $bit_{h'} := sk_j$.
- If $cmd_{h'} = C(h)$, request bit_h and let $bit_{h'} := bit_h$ if $b = 1$ and $bit_{h'} := 0^{|bit_h|}$ if $b = 0$.

– After the i-th query, $\mathcal{T}$ computes LKnow_i using the lazy knowledge algorithm.

Furthermore, the decryption oracle $\mathcal{D}$ performs the following actions:

– **If $(c, j) = (c_{\tilde{h}}, j_{\tilde{h}})$ for some $\tilde{h}$, retrieve (s, t) from $\mathcal{T}$ using the query *decrypt*$(\tilde{h})$.**

– **If $(c, j) \neq (c_{\tilde{h}}, j_{\tilde{h}})$ for all $\tilde{h}$, retrieve (s, t) from $\mathcal{T}$ using the query *xdecrypt*(c, j).**

– **If $(s, t) \neq$** $\bot$ compute

$$r := t \oplus H(s), \ (m, z) := s \oplus G(r) \text{ with } |m| = k - k_1 - k_0 \text{ and } |z| = k_1.$$

If $z = 0^{k_1}$, return m. Otherwise return $\bot$.

– Otherwise, perform the following steps:
- **Query *check*(c) from $\mathcal{P}$. If the answer is a handle $\tilde{h} \neq \bot$,** then return $\bot$.
- Let Q_r be the set of all r that have been queried from G before by someone other than $\mathcal{D}$ and of all r such that $r = r_{\tilde{h}}$ for some handle $\tilde{h}$.
- Let Q_s be the set of all s that have been queried from H before by someone other than $\mathcal{D}$.
- For all $(r, s) \in Q_r \times Q_s$, perform the following steps:
  * Let g^* denote $g_{\tilde{h}}$ or the value returned by the last query $G(r)$, respectively, and let and h^* denote the value returned by the last query $H(s)$.
  * Compute $t := r \oplus h^*$, **request pk using the query *pk*(j) from $\mathcal{P}$** and check whether $f_{pk}(s, t) = c$. If not, continue with the next pair (r, s).
  * Compute $(m, z) := s \oplus g^*$ with $|m| = k - k_1 - k_0$ and $|z| = k_1$.
  * If $z = 0^{k_1}$, return m. Otherwise return $\bot$.
- If none of the (r, s) satisfied all conditions in the loop (i.e., if the loop is left without incurring a return statement), return $\bot$.

Since the only modification was to outsource all operations involving the secret and public keys into $\mathcal{P}$, we have that

$$\Pr_{10}[\mathsf{ComputeUnknown}] = \Pr_{11}[\mathsf{ComputeUnknown}].$$

Let $\mathsf{PDBreak}$ be the event defined in Definition 10, i.e., $\mathsf{PDBreak}$ occurs if a query $check(s)$ to $\mathcal{P}$ returns $h \neq \bot$ such that no query $sk(j_h)$ or $decrypt(h)$ precedes it (where j_h is the key index j used in the query $challenge(h, j)$).

We will now show that in GAME_{11} the event $\mathsf{ComputeUnknown}$ implies the event $\mathsf{PDBreak}$. Assume therefore that in an execution of GAME_{11} the event $\mathsf{ComputeUnknown}$ occurs. By definition, this implies for some h that bit_h is requested in the i-th query such that $h \notin \mathsf{LKnow}_i$. Let the h be the handle such that bit_h is the first request satisfying this condition. The request of bit_h can have the following causes (see the construction of GAME_{11}):

(i) A query $reveal(h)$ is performed.

(ii) A query $H(s)$ is performed.

(iii) $bit_{h'}$ is requested with $cmd_{h'}$ being one of $D(j,h), C(h), F(f,h_1,\dots,h_t)$ with $h \in \{h_1,\dots,h_t\}$.

(iv) The lazy knowledge algorithm requests bit_h while computing LKnow_i.

We can exclude three of these possibilities in the present case. Case (i) is excluded because after a query *reveal*(h), we will have $h \in \mathsf{LKnow}_i$. Case (iv) is excluded since the lazy knowledge algorithm never accesses bit_h for some $h \notin \mathsf{LKnow}_i$ when computing LKnow_i. Furter, assume case (iii). Since we assumed that bit_h was the first request with $h \notin \mathsf{LKnow}_i$, it follows that $h' \in \mathsf{LKnow}_i$, otherwise the request of $bit_{h'}$ would have been an earlier one. But then $h \in \mathsf{LKnow}_i$ by Definition 11. Thus we conclude that case (ii) occurred.

The query $H(s)$ will only request bit_h in the following case: It queried *check*(s) from $\mathcal{P}$ and got a handle h' such that $cmd_{h'} = E(j,h)$ for some j. Thus in order to show that $\mathsf{PDBreak}$ occurred, it remains to show that no query *sk*(j) or *decrypt*(h') has been sent to $\mathcal{P}$.

Note that $bit_{h'}$ must have been queried earlier, since otherwise the query *challenge*(h') would not have been send to $\mathcal{P}$ and thus $\mathcal{P}$ would not have answered h' to *check*(s). Since $bit_{h'}$ has been queried earlier, we know that $h' \in \mathsf{LKnow}_i$ (again since otherwise $bit_{h'}$ would be the first request with $h' \notin \mathsf{LKnow}_i$).

Assume that *sk*(j) has been queried earlier from $\mathcal{P}$. This only happens if $bit_{\tilde{h}}$ is requested with $bit_{\tilde{h}} = K(j)$. Since $bit_{\tilde{h}}$ was requested earlier than bit_h, we have that $\tilde{h} \in \mathsf{LKnow}_i$. Thus, since $cmd_{h'} = E(j,h)$ and $h' \in \mathsf{LKnow}_i$, we have that $h \in \mathsf{LKnow}_i$ in contradiction to our choice of h. Thus *sk*(j) has not been queried before *check*(s).

Assume that *decrypt*(h') has been queried before *check*(s). There are two possible causes:

(a) $H(s')$ for some s' was queried before the present query of $H(s)$.

(b) The decryption oracle was queried (before the present query of $H(s)$) with key index $j_{h'}$ and ciphertext $c_{h'}$.

In case (a), a query of *decrypt*(h') is only executed if the query *check*(s) returned h'. However, in this case the query $H(s')$ would already have requested bit_h, contradicting the assumption that $H(s)$ was the earliest query requesting bit_h.

Consider case (b). In this case, the decryption oracle $\mathcal{D}$ has been invoked by some request for bit_{h_1} with $cmd_{h_1} = D(j_{h'}, h_2)$ and with $bit_{h_2} = c_{h'} = bit_{h'}$. By the usual argument, $h_1, h_2 \in \mathsf{LKnow}_i$. Further, above we saw that $cmd_{h'} = E(j,h)$ and that $h' \in \mathsf{LKnow}_i$. Since j_h is defined as the key index j used in the query $h' \leftarrow E(j,h)$, we have $j = j_{h'}$. From this, by the definition of the lazy knowledge algorithm, it follows that $h \in \mathsf{LKnow}_i$. This is a contradiction to the choice of h, so case (b) can also be excluded.

Thus we have that $\mathsf{PDBreak}$ occurred. Therefore $\mathsf{ComputeUnknown}$ implies $\mathsf{PDBreak}$ and we have that

$$\Pr_{11}[\mathsf{ComputeUnknown}] \le \Pr_{11}[\mathsf{PDBreak}]. \tag{11}$$

By collecting the bounds in Equations 1–11, we get

$$\Pr_1[\mathsf{Guess} \wedge \neg\mathsf{Invalid}] \le \tfrac{1}{2} + \Pr_{11}[\mathsf{PDBreak}] + \mu_1,$$

where $\mu_1 = |\frac{q_G+q_H+2q_D}{2^{k-k_0}} + \frac{q_H+q_D+(q_D+q_G+q_E)^2+q}{2^{k_1}} + \frac{q_G+q_D+q(q_G+q_D+q_E)}{2^{k_0}}|$ is negligible. Since f is partial-domain oneway, and GAME_{11} runs in polynomial time, by Lemma 1 we have that $\Pr_{11}[\mathsf{PDBreak}]$ is negligible. Thus $\Pr_1[\mathsf{Guess} \wedge \neg\mathsf{Invalid}] \le \frac{1}{2} + \mu_2$ for a negligible function μ_2 and therefore $(\mathrm{Enc}, \mathrm{Dec})$ is adKDM secure. This proves Theorem 1. □

A.2 Proof of Lemma 1

Proof. Given an adversary A against the PD-Oracle $\mathcal{P}$ we construct an adversary B against partial-domain one-wayness of the underlying function f as follows.

The machine B that implements the PD-oracle with slight changes: Let q be an upper bound on the number of queries performed by A. Then B gets as input a key pair pk^*, sk^*, values $(s^*, t^*) \in S \times T$ and a value c^*. Let j^* be the i_1-th key index that is used in A's queries, and let h^* the i_2-th handle that is used in a query of the form $challenge(h, j^*)$. Then B answers to A's queries as follows (for simplicity, if we write f_{pk}^{-1} we mean an application of the secret key sk):

- $pk(j)$: If $j = j^*$, return pk^*, otherwise return pk_j.
- $sk(j)$: If $j = j^*$, return sk^*, otherwise return sk_j.
- $challenge(h, j)$: If h has already been used, ignore this query.
  - If $h = h^*$ (and thus also $j = j^*$) then set $c_h := c^*$ and return c_h.
  - If $h \neq h^*$ then choose (s_h, t_h) uniformly from $S \times T$. Set $c_h := f_{pk_{j_h}}(s_h, t_h)$. Return c_h.
- $decrypt(h)$: If $h = h^*$, return (s^*, t^*). Otherwise return (s_h, t_h).
- $xdecrypt(c, j)$ where $(c, j) \neq (c_h, j_h)$ for all h. This is equivalent to the following:
  - If $j \neq j^*$ then check whether $f_{pk_j}^{-1}(c) = (s_h, t_h)$ for some $h \neq h^*$ or $f_{pk_{j^*}}(f_{pk_j}^{-1}(c)) = c_{h^*}$. If so, return $f_{pk_j}^{-1}(c)$. Otherwise, return $\perp$.
  - If $j = j^*$ then test if $f_{pk_j}(s_h, t_h) = c$ for any $h \neq h^*$. If such an h exists, output (s_h, t_h). Otherwise, return $\perp$.
- $check(s)$: If $s = s_h$ for some h, return the first h with $s_h = s$. If $sk(j^*)$ or $decrypt(h^*)$ has been queried, check whether $s = s^*$. If so, return h^*.

We claim that this machine B behaves identically to the PD-oracle $\mathcal{P}$ until the event $\mathsf{PDBreak}$ occurs and that A's view is independent of i_1, i_2 until the event $\mathsf{PDBreak}$ occurs (assuming that the inputs sk^*, pk^* are an honestly generated key pair, (s^*, t^*) is uniformly distributed on $S \times T$ and $c^* = f_{pk^*}(s^*, t^*)$). For the queries pk, sk, $challenge$, and $decrypt$ this is straightforward. In the case of $xdecrypt$ we distinguish two cases: For $j \neq j^*$, the check performed is equivalent to checking whether $f_{pk_j}^{-1}(c) = (s_h, t_h)$ for some $h \neq h^*$ or $f_{pk_j}^{-1}(c) = (s^*, t^*)$ and then returning h or h^*, respectively. Thus in this case the answer to the query $xdecrypt$ is the same as that the PD-oracle $\mathcal{P}$ would give. For $j = j^*$, in comparison to $\mathcal{P}$, the check whether $f_{pk_j}(s^*, t^*) = c$ is missing. However, if this check held true, we would have that $(c, j) = (c^*, j^*)$ which is excluded. To see that the query $check(s)$ gives the same answers in B and $\mathcal{P}$ until $\mathsf{PDBreak}$ occurs, note that the only case where $check(s)$ would give another answer in $\mathcal{P}$ is when $s = s^*$ but neither $sk(j^*)$ nor $decrypt(h^*)$ have been queried. However, in this case h^* would be returned in $\mathcal{P}$, thus $\mathsf{PDBreak}$ occurs.[13] So altogether, we have that B behaves identically to $\mathcal{P}$ and A's view is independent of i_1, i_2 until the event $\mathsf{PDBreak}$ occurs. By $\mathsf{PDBreak}_{i'_1, i'_2}$, denote the event that $check(s)$ is queried with $s = s_h$ where h is the i'_2-th handle used by A, and no query $sk(j_h)$ or $decrypt(h)$ has been performed

[13] In slight abuse of notation, we denote by $\mathsf{PDBreak}$ not the event that $h \neq \perp$ is returned without a query of $sk(j_h)$ or $decrypt(h)$, but that some $check(s)$ is queried such that $s = s_h$ and no query $sk(j_h)$ or $decrypt(h)$ has been performed. Since for $\mathcal{P}$ these are equivalent, it is enough to show the lemma w.r.t. this slightly changed definition.

where j_h is the i'_1-th key index used by A. Obviously, if $\mathsf{PDBreak}$ occurs, then $\mathsf{PDBreak}_{i'_1,i'_2}$ occurs for some $i'_1, i'_2 \in \{1,\dots,q\}$. Since the view of A is independent of i_1, i_2, we have that $\Pr[\mathsf{PDBreak}_{i_1,i_2}] \geq \frac{1}{q^2}\Pr[\mathsf{PDBreak}]$. So it is enough to show that $\Pr[\mathsf{PDBreak}_{i_1,i_2}] =: \varepsilon$ is negligible. Observe that in the description of B, in case of the event $\mathsf{PDBreak}_{i_1,i_2}$ the inputs sk^*, s^*, h^* are never accessed. So if we run B with the inputs sk^*, s^*, h^* set to $\perp$, $\mathsf{PDBreak}_{i_1,i_2}$ still occurs with probability at least ε. Further, $\mathsf{PDBreak}_{i_1,i_2}$ implies that $check(s)$ is called an s satisfying $f^{-1}(c^*) = \perp$. So if let B output one of the values s used in $check(s)$ queries (randomly chosen), we break the partial-domain one-wayness of f with probability at least ε/q. Thus by contradiction, ε must be negligible. Thus $\Pr[\mathsf{PDBreak}]$ is negligible in an execution of B and thus also in one of $\mathcal{P}$. □

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.
3. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.
4. M. Backes, B. Pfitzmann, and A. Scedrov. Key-dependent message security under active attacks – BRSIM/UC-soundness of symbolic encryption with key cycles. In *Proc. of 20th IEEE Computer Security Foundation Symposium (CSF)*, June 2007. Preprint on IACR ePrint 2005/421.
5. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003.
6. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology: EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323. Springer, 1992.
7. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 1997.
8. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology: ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
9. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1994.
10. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology: EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994.
11. M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient constructions. In *Advances in Cryptology: ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.
12. J. Black, P. Rogaway, and T. Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In *Proc. 9th Annual Workshop on Selected Areas in Cryptography (SAC)*, pages 62–75, 2002.
13. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology: EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.
14. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67.
15. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

16. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.
17. D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
18. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
19. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *Journal of Cryptology*, 17(2):81–104, 2004.
20. O. Goldreich. *Foundations of Cryptography – Volume 2 (Basic Applications).* Cambridge University Press, May 2004.
21. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
22. S. Halevi and H. Krawczyk. Security under key-dependent inputs. To appear in *Proc. of the 14th ACM Conference on Computer and Communications Security*, 2007. Preprint on IACR ePrint 2007/315.
23. D. Hofheinz and D. Unruh. Towards key-dependent message security in the standard model, August 2007. Preprint on IACR ePrint 2007/333.
24. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
25. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.
26. M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Society Notes, Princeton, 1996.
27. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.
28. C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1992.
29. C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.
30. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.